Pythonによるクラウドサービスを用いたシステムの構築

呉　偉，菊池　健吾

本稿ではナーススケジューリング問題のためのスケジューリングシステム開発事例を通して，クラウドサービス上に Python による Web アプリケーションを構築する手法を紹介する．サーバや OS，データベースを統合した PaaS 環境「Heroku」を利用し，Web アプリケーション向けフレームワーク Dash+Plotly を用いることで，グラフィカルな Web アプリケーションを作成する．先述の Dash のほか，数理モデリングパッケージ PuLP，ORM パッケージ SQLAlchemy など，本開発に利用したさまざまなパッケージの利用方法も適宜解説する．

キーワード：クラウドサービス，ナーススケジューリング，Heroku

1. はじめに

本稿ではナーススケジューリング問題のためのスケジューリングシステム開発の事例を通して，クラウドサービス上に Python による Web アプリケーションを構築する手法を紹介する．個人あるいは小さなスタートアップを想定開発者として，Python Web アプリケーションを低コスト・短期間で開発するためのクラウドサービス・Python パッケージの説明を行う．

2. IT システム構築環境の最新事情

2.1 クラウドサービス

クラウドサービスの台頭により，IT システムの開発環境は大きく変化している．従来の IT システム開発では，ハードウェアの調達や各種ミドルウェアの設定作業，セキュリティ対策，運用管理対応などが必要であったが，クラウドサービスを用いると，物理的な構成と各種の煩雑な設定を気にせずに必要なコンピューティングリソースを効率よく調達することができる．これにより，システム構築にかかるリソース・コストを大幅に節約することができ，個人やスタートアップなどの小規模な開発グループでも短時間で Web サービスを公開することが可能になっている．

なお，クラウドサービスは提供形態として SaaS (Software as a Service)，IaaS/HaaS (Infrastructure/Hardware as a Service)，PaaS (Platform as a Service) などが挙げられるが[1]，本稿で取り上げるのは PaaS 環境の Heroku である．PaaS とはソフトウェアを構築・稼働させる環境をインターネット経由で提供するサービスであり，以下に挙げるような Web アプリケーションのためのリソースを瞬時に稼働させることができる：

- インフラストラクチャ（サーバー，ストレージ，ネットワーク）
- Operating System (OS)
- データベース (DB)
- 開発ツール，プログラミング言語の実行環境
- その他のリソース

PaaS を利用することで，Web アプリケーションの開発，テスト，デプロイ，というサイクルを無駄なく回すことができ，スムーズなアプリケーション構築が可能となる．なお，Heroku では Python を含む多くの言語をサポートしており，Python 以外にも Node.js，Go，Ruby などの言語で同様の開発サイクルを実現できる．

2.2 Python の Web フレームワーク

Python の代表的な Web フレームワークとして Django，Bottle，Flask などが挙げられるが，本稿では Dash を使用し，より迅速に Web アプリケーションを開発することを目指す．Dash は Flask をベースに，グラフ描画，Web アプリケーション向けコンポーネントなどの機能が追加された Web フレームワークであり，グラフィカルなインターフェースを前提とした Web アプリケーションに特化した機能をもっている．

ご　い
成蹊大学理工学部
〒 180–0001 東京都武蔵野市吉祥寺北町 3–3–1
wuwei@st.seikei.ac.jp
きくち　けんご
株式会社朝日新聞社開発部
〒 104–8011 東京都中央区築地 5–3–2
kikuchi-k4@asahi.com

[1] https://ja.wikipedia.org/wiki/クラウドコンピューティング#提供サービスによる分類

表 1　3 交替制勤務表の例

ナース番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30				
	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T	W	T	/	−	e	n	+
1	e	e	/	−	−	n	n	/	/	/	−	−	e	/	/	−	−	/	/	−	−	−	e	e	/	−	e	n	n	/	/	/			
2	n	/	/	/	−	−	−	−	/	e	e	/	+	−	−	n	n	/	/	−	−	−	−	−	e	e	/	/	−	n	n	9	11	4	5	1
3	−	/	−	−	/	/	−	e	e	n	n	/	/	−	−	−	−	e	n	n	/	/	/	−	−	−	−	e	/	−	−	9	13	4	4	0
4	/	−	−	e	/	−	e	n	n	/	/	/	−	e	/	/	−	e	/	+	−	−	n	n	/	/	−	−	−	−		10	11	4	4	1
5	−	n	n	/	e	e	/	−	−	−	−	n	n	/	e	e	/	/	−	−	n	n	/	/	/	/	/	−	/	/	e	10	9	5	6	0
6	/	−	e	n	n	/	/	/	−	/	−	−	n	n	/	/	−	e	e	e	e	/	−	−	n	n	/	e	e	/		10	8	6	6	0
7	/	−	−	−	e	e	/	/	−	−	e	/	n	n	/	/	−	−	/	/	−	−	−	/	−	−	−	e	/	−		10	14	4	2	0
8	−	−	e	/	n	n	/	/	−	+	−	e	/	/	−	−	/	−	n	n	/	/	/	−	e	/	/	−	−	−	e	10	11	4	4	1
9	/	/	−	e	/	−	−	−	e	e	/	/	−	e	/	−	−	−	e	/	n	n	/	/	−	−	e	/	/	−	n	10	11	6	3	0
10	+	e	/	−	−	−	e	/	/	/	/	−	e	/	−	+	/	−	−	e	/	−	−	−	e	/	−	n	n	/		10	11	5	2	2
11	e	/	−	−	−	/	−	e	/	−	n	n	/	/	/	+	−	e	/	−	−	−	−	n	n	/	/	−	e	/		10	11	4	4	1
12	/	−	−	/	−	−	/	/	/	−	−	e	/	−	−	+	n	n	/	/	/	−	−	−	−	e	/	−	−	−		10	13	2	2	1
13	−	/	n	n	/	/	+	−	/	/	/	−	−	−	e	e	e	/	/	−	−	e	e	e	/	/	/	−	−	−		10	11	6	2	1
14	−	e	/	−	n	n	/	/	−	+	/	/	/	−	−	−	e	e	/	e	e	/	−	e	n	n	/	/	/	−	−	10	9	6	4	1
15	−	−	e	/	/	−	e	e	/	−	e	e	e	/	−	−	/	/	n	n	/	/	−	−	−	/	−	−	n	n		9	11	6	4	0
16	/	−	n	n	/	/	−	−	e	e	/	−	+	/	−	e	n	n	/	/	+	−	/	−	e	e	n	n	/	e		9	7	6	6	2
17	e	/	−	e	e	/	−	−	n	n	/	/	+	−	e	/	−	−	e	/	/	−	n	n	/	/	/	−	−	e	/	10	9	6	4	1
18	/	/	−	−	−	e	n	n	/	/	/	−	−	e	n	n	/	/	−	+	−	e	e	/	/	−	e	e	/	−		10	9	6	4	1
19	−	/	−	−	−	/	+	−	e	/	n	n	/	/	−	+	e	/	−	−	−	−	−	/	/	/	/	−	−	−	−	10	14	2	2	2
20	−	e	e	e	/	/	−	e	/	−	−	−	/	−	−	−	−	−	/	/	−	e	/	−	−	/	n	n	/	/	+	10	12	5	2	1
21	n	n	/	/	−	−	e	/	n	n	/	/	−	−	e	/	−	−	e	e	e	/	/	−	+	9	10	6	4	1						
22	+	/	−	−	−	/	−	−	−	/	e	e	/	/	−	n	n	/	/	/	/	/	−	e	e	/	−	−	−	−	/	+	10	12	4	2	2
23	−	−	/	−	e	/	n	n	/	/	/	−	−	−	e	/	/	−	e	/	/	/	−	−	−	e	/	−	+	10	13	4	2	1		
24	e	/	/	−	−	e	/	−	−	/	−	e	e	e	/	/	−	−	e	/	−	−	n	n	/	/	/	−	−	−		9	14	5	2	0
25	n	n	/	/	−	−	/	−	−	/	−	e	n	n	/	/	−	−	−	−	n	n	/	/	/	−	/	/	e	e	e	9	11	4	6	0
−	9	9	10	11	10	9	8	10	7	8	7	9	7	10	11	8	9	10	8	8	9	10	13	8	9	8	11	11	11	9						
e	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4						
n	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3						

(−：日勤; **e**：準夜勤; **n**：深夜勤; ＋：他勤務; /：やすみ)

3. ナーススケジューリング

本節では，開発するナーススケジューリングのサポートシステムについて説明する．

ナーススケジューリングとは，病棟で働くナースに対するシフト勤務表を作成する業務である．池上 [1] の調査によれば，病棟の現場において勤務表の作成には平均 6.8 時間を要し，長いときには 30 時間を費す場合もあると言われている．また，病院の師長が勤務表を作成するために，休日や勤務終了後のプライベート時間を利用する事例もよく問題視される．

日本の多くの病院では，2 交替制あるいは 3 交替制によりナースのシフト勤務が組まれている．3 交替制では，日勤，準夜勤，深夜勤というシフトが存在する．手作業で作成された 3 交替制勤務表を表 1 に示す．

ナーススケジューリングでは，以下のようなさまざまな制約条件を考慮しなければならない．

1. 各日各シフトの各ナースグループ（ベテラングループや新人グループなど）からの勤務人数を不足・超過しないようにする．
2. 勤務シフトの回数の上限と下限を守る．
3. セミナーなどの特定勤務や休み希望を達成する．
4. 禁止されるシフトの並びを避ける．

また，制約 4 では具体的に，同一シフトの連続日数の上下限を守る制約と，同一シフトが連続しない場合の間隔日数の上下限を守る制約を含む．

ナーススケジューリングについてさらに詳しく知りたい読者には，池上の書籍 [1] を勧める．

4. 利用するサービス・ミドルウェアについて

本稿では以下の開発環境のもとで，Web アプリケーションの開発を行う：

OS：macOS High Sierra 10.13.6

Python：Python 3.6.5

DB：PostgreSQL 10.4

バージョン管理システム：Git 2.17.1

Mac 用 Package Manager：Homebrew 1.7.1

また，開発に用いる主なパッケージを表 2 に示す．示したすべての Python パッケージは PyPI (Python Package Index) に登録されており，パッケージ管理システム pip を用いてインストールすることができる．

5. 開発の流れ

本節では，Heroku を用いた Web アプリケーションの開発の流れを紹介する．ここで，Heroku によって生成されるクラウド上の環境をリモート環境，実際に開発作業を行う手元の環境をローカル環境と呼ぶ．

5.1 環境設定

ローカル環境と Heroku によるリモート環境の連携は Git を通じて行う．Git がインストールされてない場合は，Homebrew で以下のコマンドを利用してインストールしておく．

```
$ brew install git
```

Heroku を利用するために，Heroku.com[2]でアカウント登録を行う．登録完了後，ターミナルから Heroku を利用できるようにするため，Heroku Command Line Interface (CLI) をインストールする：

```
$ brew install heroku/brew/heroku
```

Heroku CLI をインストールすると，以下のコマンドによりターミナル上から Heroku にログインできる．

```
$ heroku login
```

Python パッケージの管理を容易にするため，仮想環境を作成できる Virtualenv をインストールしておく：

```
$ pip install virtualenv
```

5.2 ローカル環境でのアプリケーション構築

作業ディレクトリ名を「nsp_app」とする．まずは Git の初期化と，Python 仮想環境の構築を行う：

[2] https://signup.heroku.com/dc

表 2　使用する Python パッケージ

パッケージ名	バージョン	説明
dash	0.22.0	分析系 Web アプリ構築用の Python フレームワーク
dash-core-components	0.26.0	Dash アプリ用の UI コアコンポーネントセット
dash-html-components	0.11.0	Dash アプリ用の HTML レイアウト構成コンポーネントセット
dash-renderer	0.13.0	Dash アプリのレンダリング用のパッケージ
gunicorn	2018.7.1	Unix OS 用の WSGI HTTP サーバー
plotly	3.1.0	データ可視化のパッケージ
PuLP	1.6.8	数理モデリングのパッケージ
psycopg2	2.7.5	PostgreSQL データベースのアダプタ
SQLAlchemy	1.2.10	Python の SQL ツールキットと ORM (Object-Relational Mapping)
virtualenv	16.0.0	仮想環境作成ツール

```
$ cd nsp_app
$ git init # 空の gitリポジトリを初期化する
$ virtualenv venv # 仮想環境を構築する
$ source venv/bin/activate # 新規仮想環境を有効にする
```

なお，仮想環境を無効にしたい場合は，以下のコマンドを実行する：

```
$ deactivate
```

続いて，表 2 で示しているパッケージを「pip install」コマンドでインストールする．インストールが完了したら，Python アプリのパッケージの依存関係を記述するファイル「requirements.txt」を「pip freeze」コマンドで生成する：

```
$ pip freeze > requirements.txt
```

このファイルを Heroku へのデプロイ対象に含めることで，依存関係のあるパッケージのインストールがリモート環境上で自動的に行われる．

また，Heroku で用いられる「Procfile」ファイルを以下のように作成する：

```
web: gunicorn app: server
```

「Procfile」は Heroku に対してアプリ起動のために実行するプロセスを伝えるための設定ファイルであるが，ここでは詳しく説明しない．

5.3　ローカルデータベースの準備

Heroku では，一つのアプリケーションにつき一つの無料 PostgreSQL データベースを利用できる．リモート環境でこの PostgreSQL データベースを利用することを前提とし，ローカル環境にも PostgreSQL データベースを導入しておく：

```
$ brew install postgresql
```

Mac を起動するたびに PostgreSQL が自動起動するように設定する：

```
$ brew services start postgresql
```

以下のコマンドを実行すると PostgreSQL を停止できる：

```
$ brew services stop postgresql
```

ナーススケジューリングシステム用のデータベース「nsp」を作成する：

```
$ createdb nsp
```

作成したデータベースは以下のコマンドで確認することができる：

```
$ psql -l
```

続いて，Python のオブジェクト関係マッピング (Object-Relational Mapping, ORM) である SQLAlchemy を導入する．SQLAlchemy の導入により，関係データベースに格納されたデータを Python オブジェクトとして扱うことができる．まずはデータベースに接続するための engine を作成する：

```
from sqlalchemy import create_engine

DB_URL = 'postgres://@localhost:5432/nsp'
engine = create_engine(DB_URL, echo=True)
```

関数 create_engine の入力パラメータ echo を True に設定すると，実行したすべての SQL ログが出力される．続いて，作成した engine に命令を与えるためのセッションを生成する．

表 3　Nurse テーブル

Nurse
+id: Integer
+name: String(16)
+email: String(128)

```
from sqlalchemy.orm import scoped_session,
    sessionmaker

session = scoped_session(sessionmaker(bind =
    engine))
```

本稿ではナースの基本情報を保存する Nurse テーブルを例に，SQLAlchemy によるテーブルの生成を行う．説明の都合上，各ナースが id，名前 (name) と email をもつ簡略化した Nurse テーブルを表 3 に示す．Nurse テーブルのモデルは以下のように作ることができる．(SQLAlchemy に関する詳しい説明とチュートリアルは sqlalchemy.org から確認できる[3]．)

```
from sqlalchemy.ext.declarative import
    declarative_base
from sqlalchemy import Column, Integer,
    String

Base = declarative_base()

class Nurse(Base):
    # nurseテーブルの定義
    __tablename__ = 'nsp_nurse'

    id = Column('id', Integer, primary_key=
        True)
    name = Column('name', String(16),
        nullable=False)
    email = Column('email', String(128))

Base.metadata.create_all(bind=engine)
```

4 行目の関数 `declarative_base()` は各テーブルのベースクラスを作成している．また，14 行目の `Base.metadata.create_all(bind=engine)` により，設定済みのデータベースに Base を継承しているテーブルが「CREATE TABLE」される．Nurse テーブルに対する CRUD (Create, Read, Update, Delete) 操作は以下のように行う．

```
# 生成
new_nurse = Nurse(name='Hanako')
session.add(new_nurse)
session.commit()

# 読み取り
nurses = session.query(Nurse).all()
nurse_1 = session.query(Nurse).filter(Nurse.
    id == 1).one_or_none()

# 更新
nurse_1.email = 'hanako@orsj.or.jp'
session.commit()

# 削除
session.delete(nurse_1)
session.commit()
```

ナースのスケジュールを作成するには，スケジュール期間（開始日 s_date，終了日 e_date）などを保存する対象データのテーブルや，制約条件の入力に関わるテーブルなどが必要となるが，紙面の都合上省略する．また，SQLAlchemy でテーブル間の関係構築も簡単に設定できる[4]．

5.4　Heroku へのプログラムデプロイ

Heroku アプリのデプロイとは，ローカル環境のソースコードをネットワークを通じてサーバー上にアップロードし，実行環境を構成し，Web アプリを稼働させることである．本節では，Web アプリを Heroku にデプロイする基本方法を記述する．

まずは，以下のコマンドで Heroku 上にアプリ用のスペースを作成する：

```
$ heroku create
```

「heroku create アプリ名」としてアプリ名を指定することもできる．デプロイ後のアプリは「https://アプリ名.herokuapp.com」で確認できる．指定しない場合，自動的にアプリ名が付けられる．

本稿のアプリではデータベースを利用するため，Heroku のデータベースアドオンを追加する：

```
$ heroku addons:create heroku-postgresql
$ heroku config
```

「heroku config」で取得した URL を 5.3 節の SQLAlchemy の設定で利用したローカル URL と置き換える．

```
# DB_URL = 'postgres://@localhost:5432/nsp'
DB_URL = '[heroku configで得られたURL]'
```

これで，デプロイの準備が完了した．

5.5 節から 5.7 節で紹介する実装で Web アプリがローカル環境で実装できているものとして，以下の手順でデプロイを行う：

[3] http://docs.sqlalchemy.org/en/latest/index.html

[4] http://docs.sqlalchemy.org/en/latest/orm/relationships.html


```
$ git add. # すべてのファイルをgitに追加
$ git commit -m 'Initial app nsp' # ローカル
    レポジトリにコミット
$ git push heroku master # herokuへのデプロイ
```

Heroku プラットフォームは内部で「dyno」と呼ばれるアプリ実行のための仮想コンテナをもっている．コンテナとは，コンピューティングリソース，メモリ，OS，一時的なファイルシステムを備えた軽量の独立した環境で，dyno のタイプや個数をアプリに応じて変更することにより，アプリの規模や負荷に応じたリソースを調達することができる．dyno の設定によっては Heroku の有料化が必要になるが，本稿では無料 dyno を利用するために，コンテナ数を 1 に設定している：

```
$ heroku ps:scale web=1
```

稼働できた Web アプリは，

```
$ heroku open
```

を実行することでブラウザ上で確認できる．また，プログラムを修正した場合も，

```
$ git status # 修正を確認する
$ git add .  # すべての修正をコミット対象に追
    加
$ git commit -m 'a memo of the changes' # 修
    正をローカルレポジトリにコミット
$ git push heroku master
```

で再デプロイできる．なお，以下の「logs」コマンドによりリモート環境の実行ログをリアルタイムに確認することもできる：

```
$ heroku logs --tail
```

上記の流れにより，ローカル開発，ローカルテスト，デプロイ，改善という開発サイクルが低コスト，短時間で実現できる．

5.5 Dash による Web フレームワークの作成

Dash アプリはレイアウトとインタラクティブな処理から構成される．

最初に Dash アプリのレイアウト設計について紹介する．レイアウトオブジェクトは HTML 要素や属性を生成する「Dash HTML Components (DHC)」と Dash 基本 UI コンポーネントを含む「Dash Core Components (DCC)」で構成される．一般に，Web アプリケーションのレイアウトの設計時には HTML などの知識が必要になるが，UI コンポーネントを利用する場合は，そのコンポーネントを表示するための HTML を意識する必要がない．これにより，HTML や CSS，

図 1　Dash で作成したスケジュール期間選択のレイアウト

Javascript などに精通していなくても，高度な機能をもつ UI コンポーネントを Web アプリケーション内で利用することができる．下に，スケジュール期間を選択できる Web アプリケーションの例を示す．（Dash に関する詳しい説明は Dash のホームページを参照されたい[5]．）

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from datetime import date as Dt

app = dash.Dash(__name__)
server = app.server

app.layout = html.Div([
    html.H2('ナース・スケジューラー'),
    dcc.DatePickerRange(
            id='date-range-picker',
            start_date=Dt(2017,7,1),
            end_ date=Dt(2017,7,21)
     ),
    html.Div(id='display-date')
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

この例の layout には，DHC 二つと DCC 一つが含まれている．このコードにより生成された画面を図 1 に示す．

続いて，Dash アプリにインタラクティブな処理を追加する方法を簡単な例を用いて説明する．日付選択コンポーネントで日付を選択したときに，何らかの処理をリアルタイムで実行することを考えたい．Dash では各コンポーネントに対して id を定義でき，特定 id をもつコンポーネントに対しイベント処理のための callback

[5] https://dash.plot.ly/

関数を紐づけることができる．ここでは，コンポーネント「date-range-picker」の start_date と end_date が変更されるたびに，「display-date」に指定の期間を表示するという処理を実装することにする．この場合，callback 関数は以下のようになる：

```
from dash.dependencies import Output, Input

@app.callback(Output('display-date',
    'children'), [Input('date-range-picker',
    'start_date'), Input('date-range-picker',
    'end_date')])
def show_schedule(s_date, e_date):
    return '開始日：'+s_date+' 終了日：'+
        e_date
```

図 1 の選択を行う場合，「開始日：2017-07-01 終了日：2017-07-21」が日付選択コンポーネントの下にリアルタイムで表示される．

このようにして，ナーススケジューリング問題に関してユーザーからの入力が求められる情報を，ユーザーフレンドリーな UI のもとで迷いなく入力できるように画面を作っていく．

5.6 PuLP によるモデリング

この節では，本稿のアプリケーションで扱う，ナーススケジューリング問題のモデリングと解法を解説する．

池上 [1] は 3 節で紹介したナーススケジューリング問題 (NSP) を組合せ最適化問題として定式化した．紙面の都合上，目的関数のみを示す．

記号

M：ナースの集合

N：スケジューリング対象日の集合

K：シフトの集合（例：日勤，夜勤，深夜勤，休み）

R：ナースグループの集合（グループ分けの基準は業務のスキルレベルや担当患者）

G_r：グループ r に属するナースの集合

a_{rjk}, b_{rjk}：グループ r のナースが日 j にシフト k を勤務する人数の下限と上限

変数

x_{ijk}：ナース i が日 j にシフト k を勤務する場合 1，そうでない場合 0 の 0-1 変数

$\alpha^-_{rjk}, \alpha^+_{rjk}$：グループ r のナースが日 j にシフト k を勤務する人数の不足数と超過数を表す変数

ここで，$\boldsymbol{x} = (x_{ijk})$ と $\boldsymbol{\alpha}^- = (\alpha^-_{rjk}), \boldsymbol{\alpha}^+ = (\alpha^+_{rjk})$ の関係は，以下の制約式で表されている．

$$a_{rjk} - \alpha^-_{rjk} \leq \sum_{i \in G_r} x_{ijk} \leq b_{rjk} + \alpha^+_{rjk}, \quad r \in R, j \in N, k \in K, \quad (1)$$

$$x_{ijk} \in \{0, 1\}, \quad i \in M,\ j \in N,\ k \in K, \quad (2)$$

$$\alpha^-_{rjk}, \alpha^+_{rjk} \geq 0, \quad r \in R,\ j \in N,\ k \in K. \quad (3)$$

目的関数

$$\text{minimize} \quad \sum_{r \in R} \sum_{j \in N} \sum_{k \in K} (w^-_{rjk} \alpha^-_{rjk} + w^+_{rjk} \alpha^+_{rjk}) \quad (4)$$

ここで，w^-_{rjk} と w^+_{rjk} はグループ r のナースが日 j にシフト k を勤務する人数の不足数 α^-_{rjk} と超過数 α^+_{rjk} に対するペナルティ重みを表す．

NSP のほかの制約式は本稿では省略する．

定式化した問題を PuLP を用いてモデリングする．PuLP の詳しい使用方法は久保らの書籍 [2] の 11 章を参照されたい．本稿でモデリングに必要なパラメータ（問題例）を「ins」とする．NSP で使う変数 $\boldsymbol{x}$ を以下のように生成する：

```
import pulp
from datetime import timedelta

def solve_nsp(ins):
    day_diff = ins.e_date - ins.s_date
    # 最小化問題のオブジェクトを作成する
    model = pulp.LpProblem("NSP", pulp.
        LpMinimize)
    # 変数xの定義
    suf = [(i,ins.s_date+j,k) for i in ins.
        nur_set for j in range(day_diff+1)
        for k in ins.sft_set]
    x = pulp.LpVariable.dicts("x", suf, cat=
        pulp.LpBinary)
```

同様に，変数 $\boldsymbol{\alpha}^-$ と $\boldsymbol{\alpha}^+$ を表す Python 変数 alpha_m と alpha_p も定義できる．

定義した変数を用いて，目的関数と制約条件を追加する．PuLP では，LpProblem の演算子「+=」を使って目的関数と制約条件を数理計画モデルに追加できる：

```
# 目的関数（4）
model += sum(alpha_p[grp_id,day,sft] for
    grp_id,day,sft in alpha_p) + sum(alpha_m[
    grp_id,day,sft] for grp_id,day,sft in
    alpha_m), "Obj"
# 制約条件（1）
for req in ins.grp_reqs:
    model += sum(x[nur,req.date,req.shift]
        for nur in grp_nurs[req.grp_id]) >=
        req.lb - alpha_m[req.grp_id,req.date,
        req.shift], "Con_lb{:}{:}{:}".format(
        req.grp_id,req.date,req.shift)
```

モデリングが完了した後，問題を解く（スケジューリングする）段階に入るが，PuLP では数理計画ソルバーを指定することができる．本稿では，PuLP に付属し

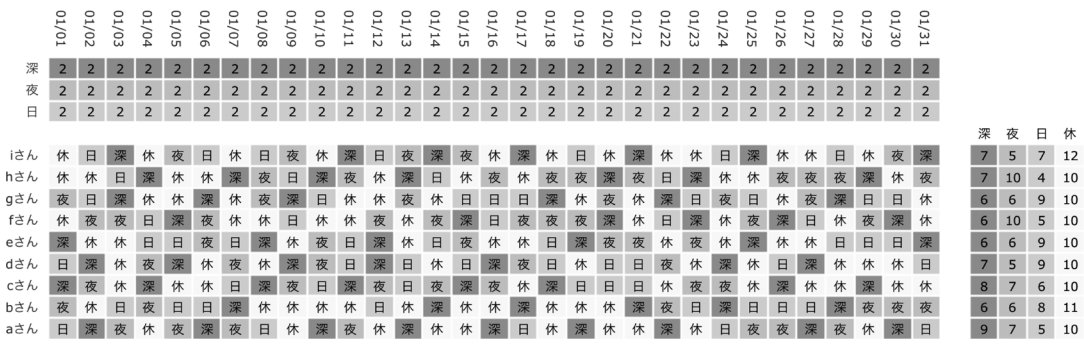

図 2　Plotly のヒートマップによる生成されたスケジュール

ている無料ソルバー CBC (COIN Branch and Cut) を使用する．LpProblem の solve 関数で問題を解く：

```
solver = pulp.solvers.PULP_CBC_CMD()
status = model.solve(solver)
```

solve 関数の戻り値（2 行目の status）で最適化の状態が確認できる．また，各変数オブジェクトの value() 関数により解（最適解あるいは暫定解）の値が取得できる．

5.7　Plotly によるスケジュール描画

本稿のアプリでは Dash と同じ企業が開発した Plotly を用いて，NSP の計算結果を可視化する．Plotly は，静的なグラフ生成などに使われる Matplotlib と比べて，よりインタラクティブなグラフが生成できるという特徴をもつ．

Plotly は多数の基本チャートをサポートしている[6]．このうち，Annotated Heatmap を用いてナースのスケジュールを描画する．Annotated Heatmap の詳しい説明と応用例は plot.ly から確認できる[7]．

紙面の都合上，描画コードの一部のみを掲載する．5.6 節で紹介した問題入力と得られた解をもとに，

- x_text：日付のリスト（1 次元 list）
- y_text：ナースのリスト（1 次元 list）
- z_val：各セルの色を表す値の配列（2 次元 list）
- z_text：各セル表示するテキストの配列（2 次元 list）
- colorscale：色と値の対応表

を設定パラメータとしてヒートマップを生成する：

```
fig = ff.create_annotated_heatmap(z_val, x=
    x_text, y=y_text, annotation_text=z_text,
     colorscale=colorscale, font_colors=['
    black'], hoverinfo='none', xgap=2, ygap
    =2)
fig.layout.margin.update({"r": 0})
fig.layout.xaxis.update({"ticks":'', "
    showgrid":False})
fig.layout.yaxis.update({"ticks":'', "
    showgrid":False})
```

また，2〜4 行のように軸のチックやマージンの設定ができる．生成した「fig」を用いて，5.5 節で紹介した DCC の Graph コンポーネントをそのまま生成できる：

```
dcc.Graph(id="schedule", figure=fig)
```

ヒートマップで可視化したスケジュールを図 2 に示す．

6.　まとめ

本稿では，Python によるクラウドサービスを用いたシステムの構築と題して，PaaS 環境の Heroku を用いた高速な開発サイクルのもとでアプリ開発を行う方法を示した．また，スケジューリングシステムの構築時に必要となるであろうさまざまな Python パッケージを紹介した．本稿が，スタッフスケジューリングシステムなどの Web アプリケーションを効率的に開発する助けとなれば望外の喜びである．

参考文献

[1] 池上敦子，『ナース・スケジューリング—問題把握とモデリング—』，近代科学社，2018.
[2] 久保幹雄，小林和博，斉藤努，並木誠，橋本英樹，『Python 言語によるビジネスアナリティクス—実務家のための最適化・統計解析・機械学習—』，近代科学社，2016.

[6] https://images.plot.ly/plotly-documentation/images/python_cheat_sheet.pdf
[7] https://plot.ly/python/annotated_heatmap/